2012-12-03

Solve problems from linear algebra with Python or R

Kuhn, Tobias Uwe

Monterey, California, Naval Postgraduate School

OA2801 Supplement

Solve Problems from Linear Algebra with Python or R

An OA2801 Supplement by Tobias Uwe Kuhn

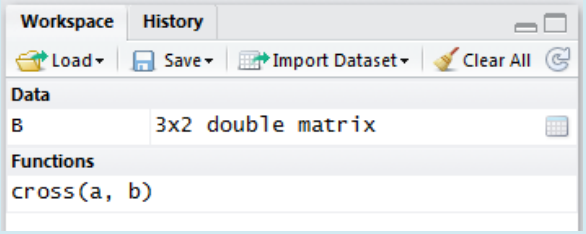
12/03/2012

1. Preface

This document is the first in a series of documents which shows how to deal with problems from Linear Algebra in Python or R. This is not meant to be another lab or exercise for OA2801 but first and foremost a guide to use Python or R as a useful tool to solve problems or verify solutions in Linear Algebra. This is also the reason why I did not focus on a particular language but gave the code for both languages we currently use in the OR curriculum. So every reader can choose the language he feels most comfortable with.

If you find any mistakes or better solutions for particular problems, feel free to contact me. I gladly enhance this document. Here are some additional notes to Python or R respectively.

Python	R

In order to use the Python code provided by this document you need to import the **`numpy`** package. Some examples also require the **`linalg`** package
 	
`> import numpy`
`> import numpy.linalg`
 	
I recommend using the following commands instead even though it is not required. However it makes your life easier.
 	
`> import numpy as np`
`> import numpy.linalg as la`
 	
With this we created an abbreviation for the packages. So instead of using **`numpy.matrix()`** or **`numpy.linalg.det()`** for example we simply use **`np.matrix()`** and **`la.det()`**.
 	
The following webpage provides additional information about the packages:
http://www.scipy.org/Tentative_NumPy_Tutorial
	I strongly recommend the use of RStudio even though it is not really required. However as with the abbreviations for Python it makes your life easier. RStudio comes with some useful tools like code completion, built-in help or the workspace browser.
	
	 
	
	With this you can easily access and edit all variables and functions of your current session. You can even create and save projects to use different workspaces. Plus many other features like plot history and management for example.
	
	This is the link to the RStudio webpage:
	http://www.rstudio.org

2. Matrix Basics

Linear Algebra	Python	R

Define Matrix A and B
$A=\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}$
$B=\begin{pmatrix}2 & 6 & 1\\ 5 & 3 & 8\end{pmatrix}$
	`>>> A=np.matrix([[1, 2],`
	`...  [3, 4]])`
	`>>> A`
	`matrix([[1, 2],`
	`  [3, 4]])`
	`>>> B=np.matrix([[2, 6, 1],`
	`...  [5, 3, 8]])`
	`>>> B`
	`matrix([[2, 6, 1],`
	`  [5, 3, 8]])`
		`> A <- matrix(c(1,3, 2,4), ncol=2)`
		`> A`
		`  [,1] [,2]`
		`[1,]  1  2`
		`[2,]  3  4`
		`> B <- matrix(c(2,6,1, 5,3,8), nrow=2,`
		`+ byrow=TRUE)`
		`> B`
		`  [,1] [,2] [,3]`
		`[1,]  2  6  1`
		`[2,]  5  3  8`

Matrix Form
$A\in\mathbb{R}^{2\times 2}$
$B\in\mathbb{R}^{2\times 3}$
	`>>> A.shape`
	`(2, 2)`
	`>>> B.shape`
	`(2, 3)`
		`> dim(A)`
		`[1] 2 2`
		`> dim(B)`
		`[1] 2 3`

Scalar Multiplication
$3A=\begin{pmatrix}3 & 6\\ 9 & 12\end{pmatrix}$
	`>>> 3*A`
	`matrix([[3, 6],`
	`  [9, 12]])`
		`> 3*A`
		`  [,1] [,2]`
		`[1,]  3  6`
		`[2,]  9  12`

Transpose
$B^T=\begin{pmatrix}2 & 5\\ 6 & 3\\ 1 & 8\end{pmatrix}$
	`>>> B.T`
	`matrix([[2, 5],`
	`  [6, 3],`
	`  [1, 8]])`
		`> t(B)`
		`  [,1] [,2]`
		`[1,]  2  5`
		`[2,]  6  3`
		`[3,]  1  8`

Inverse
$A^{-1}=\begin{pmatrix}-2 & 1\\ 1.5 & -0.5\end{pmatrix}$
	`>>> A.I`
	`matrix([[-2. ,  1. ],`
	`  [ 1.5, -0.5]])`
		`> solve(A)`
		`  [,1] [,2]`
		`[1,] -2.0  1.0`
		`[2,]  1.5 -0.5`

Determinant $\det(A)=-2$	`>>> la.det(A)` `-2.0000000000000004`	`> det(A)` `[1] -2`

Matrix Addition $A+A=\begin{pmatrix}2 & 4\\ 6 & 8\end{pmatrix}$	`>>> A+A`
	`matrix([[2, 4],`
	`[6, 8]])`
		`> A+A`
		`[,1] [,2]`
		`[1,] 2 4`
		`[2,] 6 8`

Matrix Multiplication $AB=\begin{pmatrix}12 & 12 & 17\\ 26 & 30 & 35\end{pmatrix}$	`>>> A*B`
	`matrix([[12, 12, 17],`
	`[26, 30, 35]])`
		`> A%*%B`
		`[,1] [,2] [,3]`
		`[1,] 12 12 17`
		`[2,] 26 30 35`

3. Vector Basics

Linear Algebra	Python	R

Define Vector a and b $\vec{a}=\begin{pmatrix}1 & 2 & 3\end{pmatrix}$ $\vec{b}=\begin{pmatrix}6\\ 3\\ 2\end{pmatrix}$	`>>> a=np.matrix([1,2,3])`
	`>>> a`
	`matrix([[1,2,3]])`
	`>>> b=np.matrix([6,3,2]).T`
	`>>> b`
	`matrix([[6],`
	`[3],`
	`[2]])`
		`> a <- matrix(c(1,2,3), nrow=1)`
		`> a`
		`[,1] [,2] [,3]`
		`[1,] 1 2 3`
		`> b <- matrix(c(6,3,2))`
		`> b`
		`[,1]`
		`[1,] 6`
		`[2,] 3`
		`[3,] 2`

Vector Form $\vec{a}\in\mathbb{R}^{1\times 3}$ $\vec{b}\in\mathbb{R}^{3}$	`>>> a.shape`
	`(1, 3)`
	`>>> b.shape`
	`(3, 1)`
		`> dim(a)`
		`[1] 1 3`
		`> dim(b)`
		`[1] 3 1`

Scalar Multiplication $3\vec{b}=\begin{pmatrix}18\\ 9\\ 6\end{pmatrix}$	`>>> 3*b`
	`matrix([[18],`
	`[ 9],`
	`[ 6]])`
		`> 3*b`
		`[,1]`
		`[1,] 18`
		`[2,] 9`
		`[3,] 6`

Transpose
$\vec{a}^T = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$
	`>>> a.T`
	`matrix([[1],`
	`[2],`
	`[3]])`
		`> t(a)`
		`[,1]`
		`[1,] 1`
		`[2,] 2`
		`[3,] 3`

Length of a Vector $\left\|\vec{b}\right\| = \sqrt{6^2+3^2+2^2} = 7$	`>>> np.linalg.norm(b)` `7.0`	`> norm(b, "F")` `[1] 7`

Addition of Vectors
$\vec{b}+\vec{b} = \begin{pmatrix} 12 \\ 6 \\ 4 \end{pmatrix}$
	`>>> b+b`
	`matrix([[12],`
	`[ 6],`
	`[ 4]])`
		`> b+b`
		`[,1]`
		`[1,] 12`
		`[2,] 6`
		`[3,] 4`

Dot Product
$\vec{a}\bullet\vec{a} = 14$
	`>>> np.vdot(a,a)`
	`matrix([[14]])`
	
	`>>> # have result as number:`
	`>>> numpy.vdot(a,a)[0,0]`
	`14`
		`> sum(a*a)`
		`[1] 14`

Cross Product
$\vec{a}\times\vec{b}^T = \begin{pmatrix} -5 & 16 & -9 \end{pmatrix}$
	`>>> np.cross(a,b.T)`
	`array([[-5, 16, -9]])`
	
	`>>> # have result as matrix:`
	`>>> np.matrix(numpy.cross(a,b.T))`
	`matrix([[-5, 16, -9]])`
		`> cross <- function(a, b) {`
		`+ c <- matrix(numeric(3), nrow=1)`
		`+ c[,1] <- a[,2]*b[,3]-a[,3]*b[,2]`
		`+ c[,2] <- a[,3]*b[,1]-a[,1]*b[,3]`
		`+ c[,3] <- a[,1]*b[,2]-a[,2]*b[,1]`
		`+ c`
		`+ }`
		`> cross(a,t(b))`
		`[,1] [,2] [,3]`
		`[1,] -5 16 -9`

There is no function for the cross product in R (at least I did not find it), so I provided the code to define one yourself.

4. Solving System of Equations

Linear Algebra	Python	R

Solving Ax=b [*1]
$A=\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}, \vec{b}=\begin{pmatrix}4\\ 6\end{pmatrix}$
$\left(\begin{array}{cc\|c}1 & 2 & 4\\ 3 & 4 & 6\end{array}\right)\rightarrow\left(\begin{array}{cc\|c}1 & 0 & -2\\ 0 & 1 & 3\end{array}\right)$
$\vec{x}=\begin{pmatrix}-2\\ 3\end{pmatrix}$
	`>>> A=np.matrix([[1, 2],`
	`...  [3, 4]])`
	`>>> b=np.matrix([4,6]).T`
	`>>> solve(A,b)`
	`matrix([[-2.],`
	`  [ 3.]])`
		`> A <- matrix(c(1,3, 2,4), ncol=2)`
		`> b <- matrix(c(4,6))`
		`> solve(A,b)`
		`  [,1]`
		`[1,]  -2`
		`[2,]   3`

Least Squares
$A=\begin{pmatrix}1 & 1\\ -2 & 3\\ 2 & -1\end{pmatrix}, \vec{b}=\begin{pmatrix}3\\ 1\\ 2\end{pmatrix}$
$\vec{x}^{*}=\left(A^{T}A\right)^{-1}A^{T}\vec{b}$
$\vec{x}^{*}=\begin{pmatrix}1.66\\ 1.42\end{pmatrix}$
	`>>> A=np.matrix([[ 1, 1],`
	`  [-2, 3],`
	`  [ 2,-1]])`
	`>>> b=np.matrix([3,1,2]).T`
	`>>> la.lstsq(A,b)[0]`
	`matrix([[ 1.66],`
	`  [ 1.42]])`
		`> A <- matrix(c(1,-2,2,1,3,-1),`
		`+ ncol=2)`
		`> b <- matrix(c(3,1,2))`
		`> solve((t(A)%*%A),t(A)%*%b)`
		`  [,1]`
		`[1,] 1.66`
		`[2,] 1.42`

[*1]) This only solves the system if A is nonsingular, which means it has to be of the form $\mathbb{R}^{n\times n}$ and there is a unique solution. I hope I can provide a method for systems with infinitely many solutions in the next version.

5. Eigenvalues and Eigenvectors

Linear Algebra	Python	R

Eigenvalues
$A=\begin{pmatrix}1 & 4\\ 3 & 2\end{pmatrix}$
$\lambda_1=-2,\quad \lambda_2=5$
	`>>> A=np.matrix([[1, 4],`
	`...  [3, 2]])`
	`>>> la.eig(A)[0]`
	`array([-2.,  5.])`
		`> A <- matrix(c(1,3, 4,2), ncol=2)`
		`> eigen(A)$values`
		`[1]  5 -2`
		`>`
		`> # note: reverse order`

Eigenvectors
$\vec{x}_1=\begin{pmatrix}-.8\\ .6\end{pmatrix},$
$\vec{x}_2=\begin{pmatrix}1/\sqrt{2}\\ 1/\sqrt{2}\end{pmatrix}$
Note: these are already normalized
	`>>> la.eig(A)[1]`
	`matrix([-0.8  , -0.70710678],`
	`  [ 0.6  , -0.70710678]])`
		`> eigen(A)$vectors`
		`  [,1] [,2]`
		`[1,] -0.7071068 -0.8`
		`[2,] -0.7071068  0.6`
		`>`
		`> # note: reverse order`

Both together
$\lambda_1=-2:\quad \vec{x}_1=\begin{pmatrix}-.8\\ .6\end{pmatrix}$
$\lambda_2=5:\quad \vec{x}_2=\begin{pmatrix}1/\sqrt{2}\\ 1/\sqrt{2}\end{pmatrix}$
	`>>> la.eig(A)`
	`(array([-2.,  5.],`
	` matrix([-0.8  , -0.70710678],`
	`  [ 0.6  , -0.70710678]])`
		`> eigen(A)`
		`$values`
		`[1]  5 -2`
		
		`$vectors`
		`  [,1] [,2]`
		`[1,] -0.7071068 -0.8`
		`[2,] -0.7071068  0.6`
		`>`
		`> # note: reverse order`

6. Decompositions

Linear Algebra	Python	R

QR factorization
$A=\begin{pmatrix}1 & -1 & 4\\ 1 & 4 & -2\\ 1 & 4 & 2\\ 1 & -1 & 0\end{pmatrix}$
$Q=\begin{pmatrix}-.5 & .5 & -.5\\ -.5 & -.5 & .5\\ -.5 & -.5 & -.5\\ -.5 & .5 & .5\end{pmatrix}$
$R=\begin{pmatrix}-2 & -3 & -2\\ 0 & -5 & 2\\ 0 & 0 & -4\end{pmatrix}$
	`>>> A=np.matrix([[1, -1,  4],`
	`...                  [1,  4, -2]`
	`...                  [1,  4,  2]`
	`...                  [1, -1,  0]])`
	`>>> la.qr(A)`
	`(matrix([[-0.5,  0.5, -0.5],`
	`        [-0.5, -0.5,  0.5],`
	`        [-0.5, -0.5, -0.5],`
	`        [-0.5,  0.5,  0.5]]),`
	` matrix([[-2., -3., -2.],`
	`        [ 0., -5.,  2.],`
	`        [ 0.,  0., -4.]]))`
		`> A <- matrix(c( 1, 1, 1, 1,`
		`+                -1, 4, 4,-1,`
		`+                 4,-2, 2, 0), ncol=3)`
		`> qr.Q(qr(A))`
		`     [,1] [,2] [,3]`
		`[1,] -0.5  0.5 -0.5`
		`[2,] -0.5 -0.5  0.5`
		`[3,] -0.5 -0.5 -0.5`
		`[4,] -0.5  0.5  0.5`
		`> qr.R(qr(A))`
		`     [,1] [,2] [,3]`
		`[1,]   -2   -3   -2`
		`[2,]    0   -5    2`
		`[3,]    0    0   -4`

Single Value Decomposition

$A = U\Sigma V^T$

$$A = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{pmatrix}$$

$$U = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -\frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\Sigma = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

$$V^T = \begin{pmatrix} 0 & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 1 & 0 & 0 \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

```
>>> A = np.matrix([[2, 0, 0],
...                [0, 2, 1],
...                [0, 1, 2],
...                [0, 0, 0]])
>>> la.svd(A)
(matrix([[ 0.    ,  1.,  0.    , 0.],
         [-0.707…,  0., -0.707…, 0.],
         [-0.707…,  0.,  0.707…, 0.],
         [ 0.    ,  0.,      0., 1.]]),
 array([ 3.,  2.,  1.]),
 matrix([[-0., -0.707…, -0.707…],
         [ 1.,      0.,      0.],
         [ 0., -0.707…,  0.707…]]))
>>>
>>> # note: Sigma is given by its
>>> # diagonal entries in array
```

```
> A <- matrix(c(2,0,0,0,
+               0,2,1,0,
+               0,1,2,0), ncol=3)
>> svd(A, nu=nrow(A))
$d
[1] 3 2 1

$u
           [,1] [,2]       [,3] [,4]
[1,]  0.0000000    1  0.0000000    0
[2,] -0.7071068    0 -0.7071068    0
[3,] -0.7071068    0  0.7071068    0
[4,]  0.0000000    0  0.0000000    1
$v
           [,1] [,2]       [,3]
[1,]  0.0000000    1  0.0000000
[2,] -0.7071068    0 -0.7071068
[3,] -0.7071068    0  0.7071068
>
> # note: Sigma is given by its
> # diagonal entries in $d
>
> # note: transpose last matrix for
> # A = U*Sigma*V(Transpose)
```

7. Version History

Ver2.0:

- Added Eigenvalues and Eigenvectors section
- Added Decompostions section
- Added Least Squares Method to Solving Systems of Equations section
- Replaced **`numpy`** and **`numpy.linalg`** by **`np`** and **`la`** in Python examples for better reading
- Corrected some minor mistakes

Ver 1.0:

- Created document